Interfacing the TMS320C54x DSP to an I2C Bus

Francis Kua *TI Singapore*
Yip Chan Hoe

ABSTRACT

This application report describes the way to interface the TMS320CX54x™ digital signal processor (DSP) with an I2C device. The setup uses the multichannel buffered serial port (McBSP) configured as general-purpose input/output port, and is coupled with a timer interrupt to implement the I2C protocol. An attempt has been made to provide a glueless interface to the DSP. However, due to the current driving capability of the DSP, a 74HC125 buffer is included.

A set of registers is provided to allow the user to control the I2C protocol. This particular setup has been successfully implemented and tested with the TMS320C5410 EVM from Spectrum Digital. The I2C devices used in the test are two electrically erasable programmable read-only memories (EEPROMs) (PMS3324) from Philips Semiconductor.

This document describes the DSP configured as a single master in the I2C standard mode. The hardware interface and software control of the I2C data transfer are presented. In addition, the test methodology with I2C devices is discussed. Finally, limitations and suggestions of this application are stated.

Contents

List of Figures

List of Tables

1 Introduction

There is no dedicated I2C hardware in TI's TMS320C5000™ platform of DSPs. This application report is focused on interfacing the TMS320C54x DSP as a single master to the I2C bus in standard I2C mode. With this application report, the user can extend interfaces of the TMS320C54x DSP to I2C devices.

It would be nice to provide a glueless interface to the I2C bus. However, due to the currently driving capability of the DSP, a relatively inexpensive buffer has been added between the DSP and the I2C devices. In addition, a timer and a McBSP must be allocated to implement this protocol.

1.1 Objective

The objective of this application is to build an I2C interface with a minimum hardware requirement, and a user-friendly I2C application program. This program should operate in a way similar to a device driver, and should operate in the background. This will allow actual application running at the foreground, while the I2C protocol runs at the background. A simple user-friendly protocol should be presented to the user.

1.2 Standard I2C Bus Protocol

An I2C bus [1] consists of two bidirectional lines: serial data line (SDA) and serial clock line (SCL). The output of I2C devices is open-drain, or open-collector, in order to perform the wired-AND operation. The arbitration procedure between multi-master relies on the wired-AND connection to prevent data corruption. The master in control of the bus always generates the clock signal in SCL, while the slower I2C slave may stretch the clock by holding down the clock line. Thus, synchronization is achieved among devices with different clock speeds.

Every element of data transmitted or received is 8 bits long, with the most significant bit (MSB) transmitted first. The I2C protocol requires the data on the SDA to be stable during the HIGH period of SCL. Transition on the SDA line can only take place while the clock is low. Otherwise, this would be recognized as a start or stop condition. The number of bytes per transfer is unrestricted, but each byte must be followed by an acknowledgement bit from the slave. Hence, whenever the master sends a byte, it will check for a acknowledgement bit (low at SDA) from the slave. In the event where an acknowledgement is absent, an error condition occurs.

As the I2C protocol allows multiple slave devices to be connected to a single master, a form of addressing must be present. This address information is stored at the very first byte of the data sent. Only 7 bits are used to address the slave device, while the last bit signifies whether a read or write in initiated by the master. Figure 1 shows a timing diagram of the master receiving one byte from the slave. To put a logic high onto the I2C bus, the I2C device **releases** the line where it is connected to the pullup resistors. On the other hand, to put a logic low onto the bus, the device **pulls** the line low.

TMS320C5000 is a trademark of Texas Instruments.

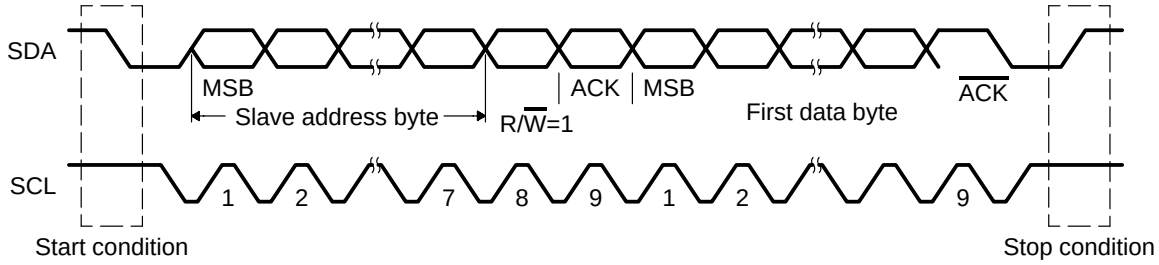


Figure 1. Receiving One Byte From the I2C Slave

Figure 2 shows the internal connection of I2C slave devices. Notice that the transistors used are open-drain. This open-drain characteristic allows bidirectional Iinput/output (/O) over the two signal lines.

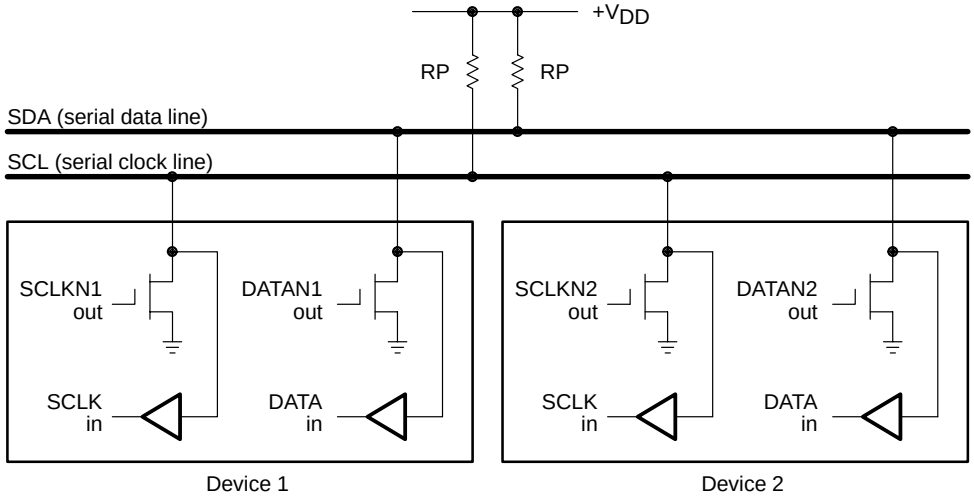


Figure 2. Connection of I2C-Bus Devices to an I2C Bus

1.2.1 Open-Drain Output Characteristic

All I2C devices have open-drain output terminals. I2C data transfer is based on this characteristic for proper operation, particularly the synchronization and arbitration procedure. See Figure 3.

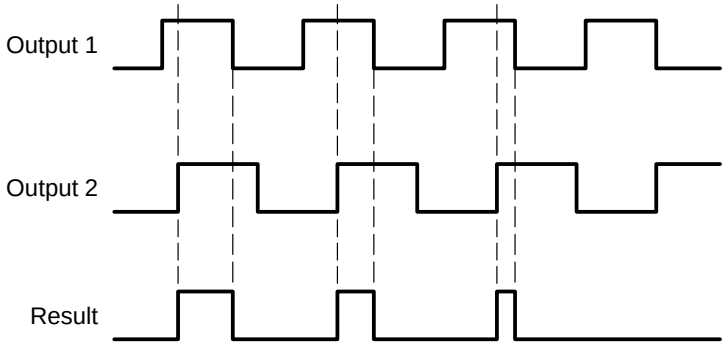


Figure 3. Example of Two Open-Drain Outputs wire-AND Together

The TMS320C54x general-purpose input/output (GPIO) is not bidirectional. Hence, we have to use 2 GPIOs each to connect to SDA and SCL. One of the GPIOs is configured as input, while the other is configured as output.

1.2.2 I2C Features Supported

This application report focuses on the simple data transfer between the TM320C54x (master) and slave devices. The main features of I2C supported are:

- Single master, multi slaves
- Up to 100 Kbit/s
- Clock synchronization with slower slave
- 7-bit addressing

2 I2C Interface: McBSP as GPIO

The choices for the input/output (I/O) ports on the DSP include the host port interface (HPI) and the multichannel buffered serial port (McBSP). The latter is preferred, as there is normally more than one McBSP, and only one HPI port available.

In this setup, the McBSP port is configured as a general-purpose I/O port (GPIO), in which four lines are used (see Table 1). FSX and DX are used as input and output pins for the SDA line, while DR and FSR are used as input and output pins for the SCL line. Four pins are used to interface to SDA and SCL because the GPIOs are not bidirectional, as mentioned earlier. The circuit of the I2C interface is shown in Figure 4.

A non-inverting line buffer, 74HC125, is used to provide the current driving capability and isolation to the TM320C54x. Pullup resistors [1], R_P, with 10Kilo ohms are used for the slave devices, while series resistor, R_S, is used to accommodate voltage drop if there is logic difference.

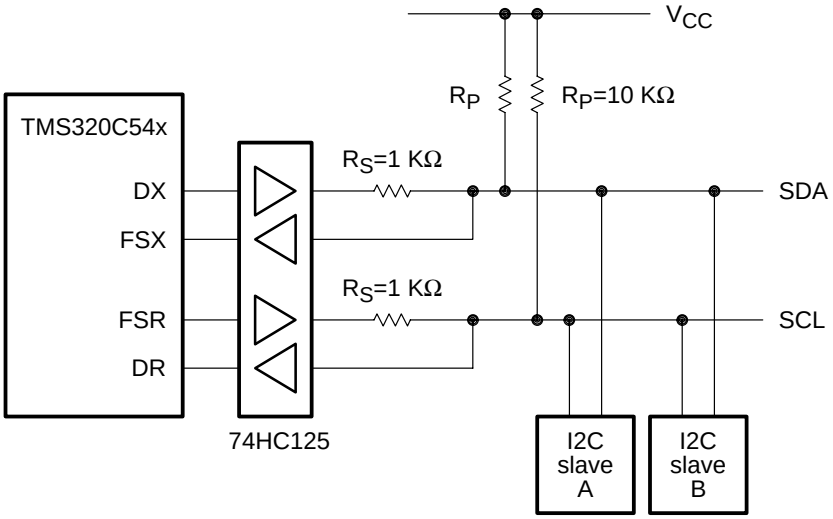


Figure 4. Hardware Setup for I2C Bus Interface

The difference in voltage level occurs when output of the buffer is high, but the I2C slave device may pull the line low (see Figure 5 for more examples). This will create a short-circuit path between the buffer and the I2C devices. To circumvent this, a series resistor of a suitable value is used to limit the current. The calculation of the resistance values are discussed in section 6.2.

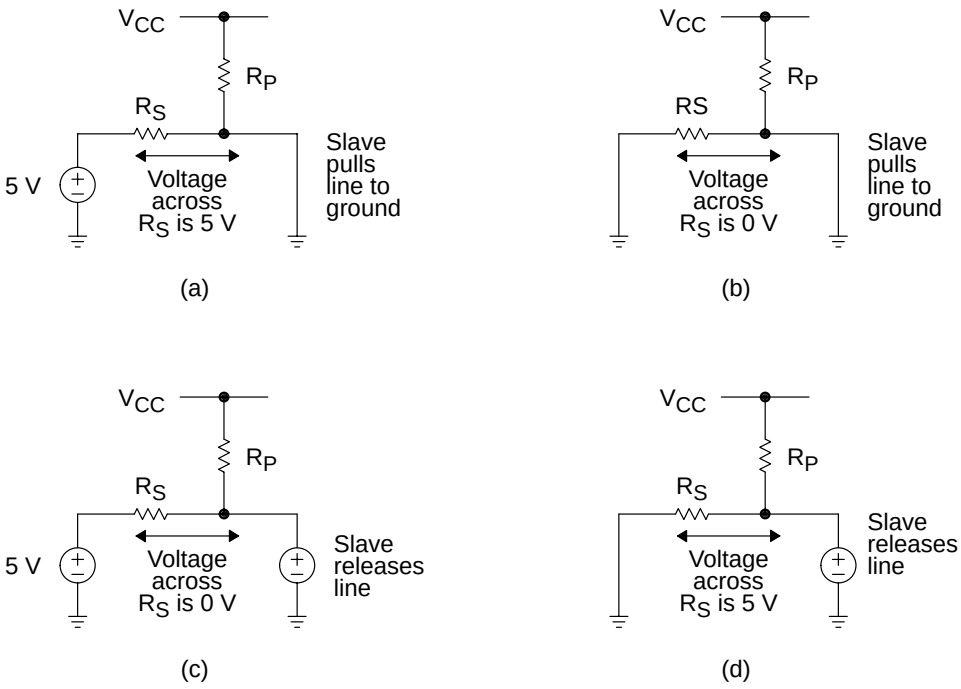


Figure 5. Four Situations Arise on the Need of Series Resistor, R_S

2.1 McBSP Configuration for GPIO

The serial port control register 1 and 2 (SPCR 1 and 2) and pin control register (PCR) are configured so that the McBSP is to be used as a general-purpose I/O, rather than a serial port. The bit configuration is shown in Table 1.

Table 1. Configuration of Pins as a General-Purpose I/O

Pin	GPIO Enabled by Setting Both:	Selected as Output:	Output Value Driven From:	Selected as Input:	Input Value Readable on:
DX	XRST* = 0 XIOEN = 1	Always	DX_STAT		
FSX	XRST* = 0 XIOEN = 1			FSXM = 0	FSXP
DR	RRST* = 0 RIOEN = 1			Always	DR_STAT
FSR	RRST* = 0 RIOEN = 1	FSRM = 1	FSRP		

The initialization routine is written as a subroutine, *_init_gpio*, in the source file, *i2c.asm*. This routine can be called from C or the assembly program.

3 Timer Interrupt

The I2C application monitors SCL and SDA lines periodically by the use of a timer interrupt. It is critical to get an optimum sample period, while not missing a valid signal transition.

From the timing specifications of the I2C, the transfer rate is 100 Kbit/s, or 10 μs (SCL clock period). This can be derived from adding t_{HIGH}, t_{LOW}, t_r and t_f. See Figure 6.

- CLK high pulse width, t_{HIGH} = 4 μs
- CLK low pulse width, t_{LOW} = 4.7 μs
- CLK rise time, t_r = 1 μs
- CLK fall time, t_f =0.3 μs

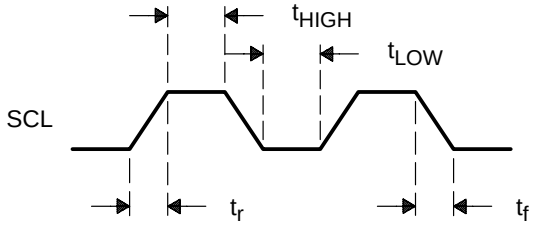


Figure 6. Timing Characteristics of Clock (SCL)

The time for the periodic sample should be less than 4 μs. This application report uses a 1-μs timer interrupt to poll for the status of the SCL line.

Again, from the timing specification of I2C:

- Hold time for start condition, $t_{HD;STA}$ = 4 μs
- Setup time for start condition, $t_{SU;STA}$ = 4.7 μs
- Data hold time, $t_{HD;DAT}$ = 0
- Data setup time, $t_{SU;DAT}$ = 250 ns
- Setup time for stop condition $t_{SU;STO}$ = 4 μs

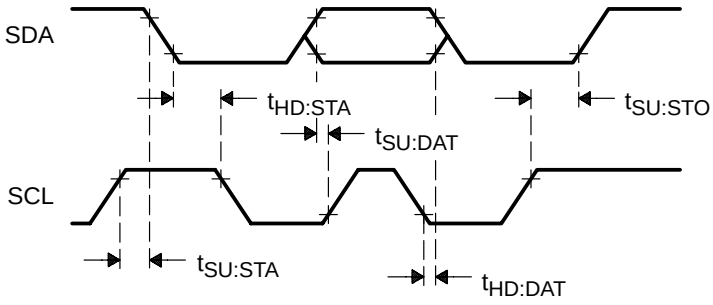


Figure 7. Timing Characteristics of SDA a nd SCL

To satisfy all the timing requirements and program simplicity, a common 5-μs timing is chosen for the timer interrupt for the five cases (separate timing can be adjusted in the definition section of the source file. See Table 3). In addition, a 1-μs interrupt is also used to check for signal transition.

4 Usage and Description of the I2C Program

The software design methodology involves the following considerations:

1. An I2C timer interrupt service routine (ISR) to be operated in the background
2. The user only needs to specify in the main program which is the mode of data transfer, where the data is stored, and how long he data is.
3. The I2C ISR will be transparent to the user, and when transmission is over, it will signal the user by setting the **SUCC** flag in the **I2CSTATUS** register.

From this section onward, the I2C terms and definitions used are described in Table 2.

Table 2. I2C Terms and Definitions

Term	Description
Transmitter	The device which sends data to the bus.
Receiver	The device which receives data from the bus.
Master	The device which initiates a transfer, generates clock signal, and terminates a transfer. TM320C54x is a single master in this setup.
Slave	The device addressed by the master. The number of devices that can be connected is limited by the bus capacitance of 400 pF. Above this value, there will be excessive time delay and hence, lower transfer rate.
Synchronization	Procedure to determine one clock signal among devices with different clock speeds. The low period of SCL is determined by the device with the longest low clock period. The high period of SCL is determined by the device with the shortest high clock period.
Start condition	A high-to low-transition on the SDA line while SCL is high. The master will control the bus after this.
Stop condition	A low-to-high transition on the SDA line while SCL is high. The bus will be free after this.
Acknowledge bit	A receiver sends an ACK bit or ACK* bit after a byte has been received. • In master-transmitter mode, the slave acknowledges by pulling down SDA so that it remains stable-low during the high period of the 9th clock pulse. • In master-receiver mode, the operation is the same as above except during the first byte and the last byte. In the first byte, it is the sending slave address, and it operates like a master-transmitter. In the last byte, the master signals the end of data to the slave transmitter by not generating an ACK (or generating an ACK*) before the stop condition.

4.1 I2C Functions

A total of five functions, listed in Table 3, are provided to implement the I2C operation.

Table 3. Driver Functions

Function Name	Description
_init_gpio	Initializes McBSP as GPIO.
_init_i2c	Initializes I2C control registers, the timer interrupt mask register (IMR), and the period to interrupt is set. All byte/bit counters are initialized, and the slave address is loaded. It is necessary to call this subroutine prior to any transmit or receive function.
_i2c_write	Initializes master-tansmitter mode, and data transmission to slave devices starts.
_i2c_read	Initializes master-receiver mode, and data reception from slave devices starts.
_USER_FUNCTION	This function is called when data transfer is over. Here, it serves as a good control sequence for the user for the next transfer.

NOTE: All functions are C-callable.

4.2 Other Registers: Definitions and Initializations

The memory registers allocated in the I2C ISR are used to control the flow of program and set up the correct bit for transfer or storage and error report. They are listed in Table 4, and the complete description can be found in Appendix A.

Table 4. Control Registers Used in I2C ISR

Register Name	Description
_I2CSTATUS†	I2C status register. Controls the mode and flow of operation.
_ODPTR†	The address of the first byte in the output data (total 256 bytes) memory location.
_ODBYTECTR†	Output data byte counter. The first 8 bits contain the pointer to the current outgoing byte, and the next 8 bits are used to store the total number of bytes to send. Only 255 bytes can be sent at a time.
ODBITCTR	Output data bit counter. Contains the counter to the current outgoing bit.
IDPTR	The address of the first byte in the input data (total 256 bytes) memory location.
_IDBYTECTR†	Input data byte counter. The first 8 bits contain the pointer to the current incoming byte, and next 8 bits are used to store total number of bytes to receive. Only 255 bytes can be received at a time.
IDBITCTR	Input data bit counter. Contains the counter to the current incoming bit.
ERRORCODE	Contains the type of error occurred during I2C ISR.
_SLAVE_ADDR†	Contains the 7-bit slave address (receiver or transmitter). Defined by the constant, K_SALVEADDR, in *i2c.inc* include file.

† These are C-addressable memory registers.

The initialization of I2C data transfer protocol is written in a call subroutine, *init_i2c* in *i2c.asm*, in which the timer interrupt mask register (IMR) and period to interrupt is set, all byte/bit counters are initialized, and the slave address is loaded. These registers can be modified to the programmer's preference after this subroutine is called. It is necessary to call this subroutine prior to any transmit or receive function (see example below).

```
.ref  _init_gpio,_init_i2c    ;Reference from i2c.asm.
                              ;
      ,, ,,                   ;
CALL  _init_gpio              ;Go to _init_gpio subroutine.
CALL  _init_i2c               ;Go to _init_i2c subroutine.
                              ;Start operation INTM are enabled.
```

To disable I^{2}C transfer protocol after the start of timer interrupt (see the two examples below):

```
ANDM  #K_DISABLE_TINT, IMR    ;Clear TINT maskable register.


LD     #_I2CSTATUS, DP        ;Set up DP.
ST     #K_DISA_0, _I2CSTATUS  ;Clear i2c disable flag in
                              ;_I2CSTATUS(K_DISA_0 & I2C_DP,
                              ;defined in i2c.inc include
                              ;file).
```

4.3 I2C Operation Mode

This section describes the necessary steps for configuring the mode of a I2C operation, i.e., master-transmitter or master-receiver.

Master-Transmitter (MT)

Two registers, _**ODPTR** and _**ODBYTECTR,** must be defined before calling the subroutine, *i2c_write*, for sending data to the slave. Although TMS320C54x is a 16-bit device, only 8 bits of a memory word are used.

Address	_ODPTR	_ODPTR+1	_ODPTR+2	~ ~	_ODPTR+254	_ODPTR+255
Content	Slave address	1st data byte	2nd data byte	~ ~	254th data byte	255th data byte

	15 … 8	7 … 0
_ODBYTECTR	0	Number of bytes to send

The actual address for data starts at _**ODPTR+1**, and the number of bytes to send must be defined in the _**ODBYTECTR** register. After this, a call to ***_i2c_write*** subroutine will initialize the _**I2CSTATUS**, and shift the slave address 1 bit to the left, to append transfer direction (R/W*), i.e., 0 for write, to the least significant bit (LSB) of the first byte (slave address byte).

To send 11H, 33H, and 55H to the slave (default address), see the example below:

```
STM         #_ODPTR+1,AR1           ;AR1 points to first data byte loc.
ST               #11H, *AR1+        ;11H store to ODPTR+1
ST               #33H, *AR1+        ;33H store to ODPTR+2
ST               #55H, *AR1+        ;55H store to ODPTR+3
ST               #3, _ODBYTECTR     ;3 bytes to send
CALL  _i2c_write                    ;Initialize write operation
                                    ;i2c transfer starts from here
```

To send 100 bytes addressed by AR2 to the slave (address A0H), see the example below:

```
STM         #_ODPTR+1,AR1           ;AR1 points to first data byte loc.
RPT         #99                     ;Repeat next instruction for 100 times
MVDD  *AR2+, *AR1+                  ;Move data to ODPTR memory
ST               #100,_ODBYTECTR    ;100 bytes to send
ST               #A0H,_SLAVE_ADDR   ;Overwrite default slave address
CALL  _i2c_write                    ;Initialize write operation
                                    ;i2c transfer starts from here
```

Master-Receiver (MR)

Only one register, _IDBYTECTR, must be defined before calling the subroutine, *i2c_read*, for receiving data from the slave (lower 8-bit, while the higher 8-bit is clear). This register will define how many bytes the master should receive. The definition of the input data register is illustrated below.

Address	_IDPTR	_IDPTR+1	_IDPTR+2	~ ~	_IDPTR+254	_IDPTR+255
Content	Slave address	1st data byte	2nd data byte	~ ~	254th data byte	255th data byte

	15 ... 8	7 ... 0
_IDBYTECTR	0	Number of bytes to receive

The actual address for data starts at _IDPTR+1, and the number of bytes to receive, must be defined in the _**IDBYTECTR** register. After this, a call to the _***i2c_read*** subroutine will initialize the _**I2CSTATUS**, and shift the slave address 1 bit to the left, to append transfer direction (R/W*), i.e., 1 for read, to the LSB of the first byte (slave address byte).

To receive 20 bytes from slave (address: 44H), see the example below:

```
ST    #20, _IDBYTECTR   ;20 bytes to receive
ST    #44H, _SLAVE_ADDR ;Overwrite default slave address
CALL  _i2c_read         ;Initialize read operation
                        ;i2c transfer starts from here
```

5 Software Flow

This section explains the software process flow of the I2C operation implemented. The flow chart of the application are illustrated in Appendix B.

5.1 Main Control Loop

The flow chart shown in Figure D–11 shows the main function of the I2C Timer interrupt service routine. The flow of the functions is sorted in order of priority, and one process is executed per timer interrupt. The _I2CSTATUS register is updated in the current cycle, and the subsequent timer interrupt will be based on it.

The main loop will first check if I2C data transfer is a disabled, successful, or erroneous transfer. If any of the above three cases is true, the timer interrupt will be disabled. It also consists of the following sequences: release SCL, PULL SCL acknowledgement bit, start condition, stop condition, master-receiver, and master-transmitter. At the end, it will enable the timer interrupt again, and the whole cycle continues.

5.2 SCL Goes High Sequence

This sequence has the highest precedence in the loop because of the need to generate the clock signal for data transfer. This is done if the SCLH flag of _I2CSTATUS is set.

There are basically three tasks in this sequence, i.e., synchronization, receive data in MR mode, and clock out data in MT mode. In the synchronization task, SCL is released and is monitored. If it does not maintain at its high level, that means some other device is holding down the SCL line. The whole process is repeated in the subsequent interrupt. This allows the master to synchronize with the slower slave. For the other two tasks, it is mainly generating the clock to receive or send data.

The period of the next timer interrupt (defined in K_SCL_HIGH pulse width) and SCL goes low flag, CLKL, are set so that one clock pulse is generated.

5.3 SCL-Goes-Low Sequence

No synchronization is needed in this sequence, due to the nature of the open-drain output of the I2C device. SCL will always pull low, no matter what logic level is in other devices.

5.4 Acknowledge-Bit Sequence

In MR mode, the first byte (slave address) to be sent is handled by MT mode. For the rest of the data, the master pulls SDA as an acknowledgement to the slave during this bit. It is important to release this line at the end of the acknowledge bit. Following I2C specification, the master will not pull SDA (ACK*) for the last byte before the stop condition.

In MT mode, the SDA line is released for the slave to acknowledge through this line. SCL will go high in the next interrupt, and th eACK bit will be checked by the master within the SCL-goes-high sequence.

5.5 Start-Condition Sequence

The start condition begins when both SDA and SCL are high. SDA is then pulled low, and the next interrupt is set for the period of $t_{HD;STA.}$ When the interrupt is generated, SCL is pulled low, thus completing the start condition.

5.6 Stop-Condition Sequence

Before beginning the stop condition, SCL is released, and SDA is pulled low. After an timer interruption of $t_{SU:STO}$ long, SDA is released. This marks the end of the stop condition.

5.7 Master-Receiver Mode Sequence

Two main functions involved in this sequence are: sending the slave address byte and receiving data from when SCL is high. Prior to receive data, the master must specify which slave is to be addressed. Hence, for the first byte, it is sending the slave address through the MT mode sequence. After that, it will read th eSDA line with every clock.

5.8 Master-Transmitter Mode Sequence

The master sends out data onto the SDA line prior to generating the clock on SCL line. This cycle continues for 8 times (byte) until it frees the SDA line for acknowledgement from the slave.

Figure 8 shows the timing diagram of a master sending a byte to a slave device of address 58H. The vertical lines signify the timer interrupts, which are defined in Table 3.

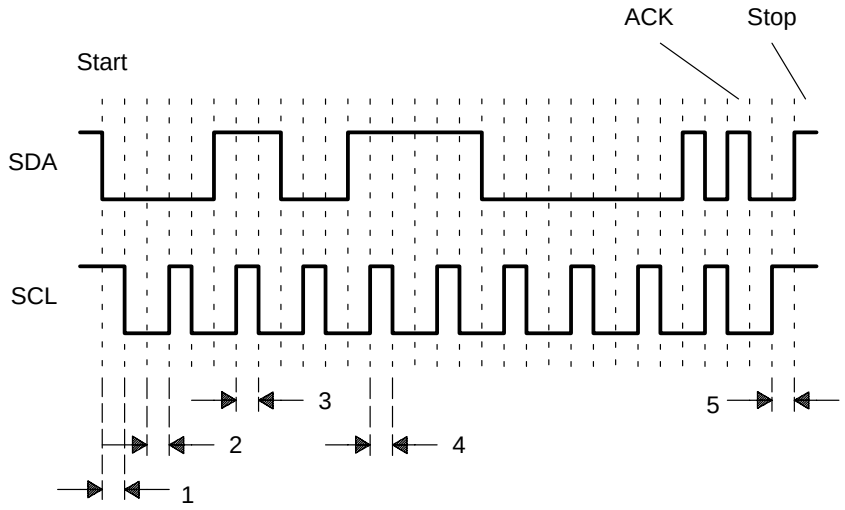


Figure 8. Master Sending Byte to Slave Device of Address 58H

As shown in Figure 8, the single-master I2C device, i.e., the C54x DSP, starts off only when the I2C-bus is high. When a data transmission is initiated, the master will execute a start condition by pulling SDA while SCL is high. As shown in Figure 7, the time to observe during the start condition is $t_{HD;STA}$ (hold time for the start condition). Hence, the time to interrupt will set at K_START_HOLD_TIME. After this interrupt, the program will pull SCL low, thus completing the start-condition sequence.

In Figure 6, the SCL low pulse width, t_{LOW}, is 4.7 μs. Hence, the time to the next interrupt will set at K_SCL_LOW2. This will meet the minimum low pulse width of the I2C clock. When the interrupt comes, this is the time to output data onto SDA line. This instant is shown at the third vertical fine line in Figure 8, in which the data is zero. In Figure 7, there is a data setup time, $t_{SU;DAT}$, to satisfy. Hence, the time to interrupt will set to K_SCL_LOW1. Note that $t_{SU;DAT}$ is 250 ns from the specification, while the default time is set at 5 μs in the program. After this interrupt, the master will release the SCL line. This is the point where the synchronization process starts. The master will check if SCL went high. If it did not go high, the interrupt will be set at 1 μs, and will continuously poll for SCL at this interval until the slave releases SCL high. This scenario is depicted in Figure 9.

If the slave releases SCL, the clock high pulse width has to be defined. In Figure 6, the SCL high pulse width, t_{HIGH} is 4 μs. Hence, the time to the next interrupt will be set at K_SCL_HIGH. SCL will remain high for the K_SCL_HIGH microsecond, and the master will pull SCL low when the next interrupt comes. Since I2C has a wired-AND open-drain output characteristic, SCL will go low, regardless of the logic level of the slave devices. The same cycle continues when SCL goes low again for the K_SCL_LOW2 microsecond. After which, th emaster will output data to the SDA. This instant is shown at the sixth vertical fine line in Figure 8.

For master-receiver mode, the operation is similar to master-transmitter mode, as described earlier. The master will generate clock pulses onto SCL line and perform any clock synchronization as necessary, if the slave devices slow down the transmission by stretching the low pulse width of SCL. The master will read from the SDA line during the high pulse of the SCL.

The timing diagram of the data transfer (without slave devices) is measured with an oscilloscope, as shown in Appendix C. All the parameters in Table 3 satisfy the I2C specification.

Table 5. Timing of SCL Defined in the Program Include File (time.inc)

No.	I2C Timing Spec. Terms	Definitions in Source Code	Default Duration (μs)
1	$t_{HD;STA}$	K_START_HOLD_TIME	5
2	–	K_SCL_LOW1	5
3	–	K_SCL_LOW2	5
4	–	K_SCL_HIGH	5
5	$t_{SU;STO}$	K_STOP_SETUP_TIME	5

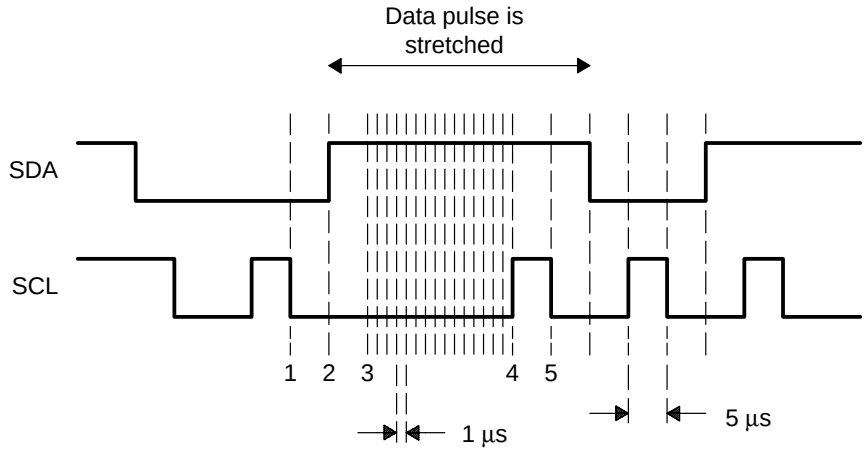


Figure 9. Example of Synchronization of SCL in the Master-Transmitter Mode

In Figure 9, the scenario of using a slower I2C slave device is illustrated. Normal clocking and data writing ends at the third instant in Figure 9 when the master releases SCL but the SCL line does not go high. This happens frequently in slower I2C devices where more time is needed to process the previous data bit or byte. An example is a memory device (EEPROM), where there is an inherent write/erase cycle between received data, thus prolonging the data transmission. This is also the reason why an EEPROM is used to test this application in section 6.

Since SCL does not go high, a shorter time to poll for the change is needed. In this application, a default time of 1 µs is used. This is illustrated between the third instant and the fourth instant. There are multiples polling at the 1-µs interval for the slave to release SCL until the fourth instant where SCL finally goes high. Then, normal operation resumes where the K_SCL_HIGH microsecond is initialized for the next interrupt. This results in the interrupt in the fifth instant shown in Figure 9.

The quick polling time is suggested at 1 µs. Although the user can reduce this time for better and faster monitoring of any logic change in SCL, it is necessary to consider the instruction cycles of the user's main application and this I2C application. In addition, it is necessary to bear in mind that I2C is not meant for fast data communication, but for its flexibility in addressing and controlling I2C slave devices.

6 I2C Data Transfer Test

The I2C data transfer is verified with a **PCF8582 I2C EEPROM** [2] from Philips Semiconductor. The objective of this test is to prove the validity of I2C functions and the integrity of data transfer. The test involved writing some data into the EEPROM, and checking the content by reading them back.

6.1 PCF8582 EEPROM With I2C-Bus Interface

PCF8582 is a floating-gate EEPROM with 256 × 8 bits. The device type identifier is fixed at **1010B**, and these are the 4 MSBs of the 7-bit device address. Chip select is done by three address inputs (A_2, A_1, A_0), and up to eight similar devices can be connected to the I2C bus. These three address inputs make up the rest of the LSB of the device address. The memory address to the 256 locations is called "word address", starting from 0 through 255. See Figure 10.

MSB							LSB
1	0	1	0	A2	A1	A0	R/W*

Figure 10. Device Address and Operation Mode

6.2 Hardware Setup

The I2C interface circuit is setup as shown in Appendix D. The power supply, V_{CC}, is 5 V. The output high current, I_{OH}, of the buffer [3] is in the range of 6 mA and the typical output high voltage, V_{OH}, is 4.3 V. When there is a logic difference across series resistor, R_S, the resistance is calculated to be around 700 Ω. Therefore, R_S is chosen to be 1 KΩ.

The total maximum input high current of the two I2C devices are 20μA. The pullup resistors, R_P, are chosen to be 10kΩ from figures 25 and 28 of the I2C specification sheet [1]. Note that R_S, mentioned above, is not the series protection resistor explained in the I2C specification sheet.

The chip select pins (A_2, A_1, A_0) of EEPROMs are set to 000 and 001. Therefore, Slave 1 and Slave 2 addresses are 50H and 51H, respectively.

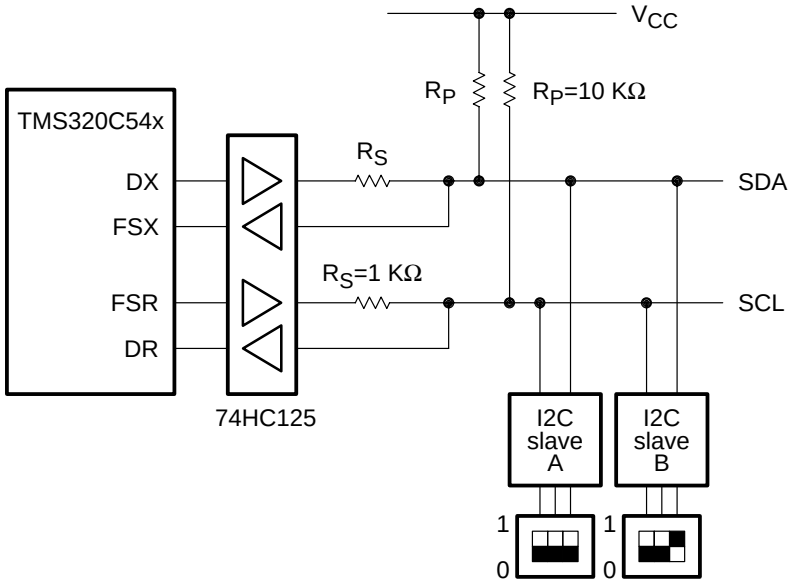


Figure 11. I2C Bus Interface With EEPROMs

6.3 Test Methodology

The following steps are carried to test the I2C data transfer on Slave A.

- Send slave address and write operation bit: 1010 0000 (A0H).
- Send word address: 0 (write data from memory address 0 and onward).
- Send some bytes of data to EEPROM (check acknowledge from EEPROM).
- Wait for EEPROM erase/write (E/W) cycle [2].
- Send slave address and write operation bit: 1010 0000 (A0H).
- Send word address: 0 (EEPROM points back to memory address 0).
- Send slave address and read operation bit: 1010 0001 (A1H).
- Read the same number of bytes of data from EEPROM.

I2C data transfer is successful if data is written to EEPROM with acknowledgement and the same data is read back.

7 Conclusion

An I2C data transfer driver is developed on the TMS320C54x DSP using a timer and McBSP. The application has been tested on EEPROM from Philips Semiconductor with the DSP configured as a master. An easy and user-friendly interface has been defined to allow the user to make use of these functions with minimum effort. In addition, this I2C application runs in the background, and will call a user function once the I2C operation is complete. This allows the user to run their main application, while accessing the I2C devices at the same time.

8 References

1. The I^2C-bus and how to use it, Phillips Semiconductor, April 1995.
2. PCF8582 256×8 bit CMOS EEPROM with I^2C-bus interface, Phillips Semiconductor, Feb 1997.
3. *SN54HC125, SN74HC125 Quadruple Bus Buffer Gates With 3-State Outputs*, (SCLS104B).

Appendix A Memory Registers

A.1 I2CSTATUS

15	14	13	12	11	10	9	8
DISA	–	SUCC	ERR	ACK	STOP	CLKL	CLKH

7	6	5	4	3	2	1	0
START	MODE		ADDR	PRESCL	PRESDA	CURSCL	CURSDA

Bit	Name	Function
15	DISA	Disables the I2C data transfer operation.
14	–	Not used. (For user's application.)
13	SUCC	Success in sending or receiving data.
12	ERR	Error in sending or receiving data. Type of error reported in ERRORMSG.
11	ACK	Indicates current bit is acknowledge from master or slave
10	STOP	Indicate current byte is the last data to send or receive
9	CLKL	SCL goes LOW flag
8	CLKH	SCL goes HIGH flag
7	START	Initiates start condition
6, 5	MODE	10: Master–receiver mode 01: Master–transmitter mode
4	ADDR	Indicates current byte is slave address
3	PRE SCL	Previous SCL logic level
2	PRE SDA	Previous SDA logic level
1	CUR SCL	Current SCL logic level
0	CUR SDA	Current SDA logic level

A.2 ERRORMSG

15			3	2	0
x					ERRCODE

Bit	Name	Function
15 – 3	–	Not used
2, 1, 0	ERRCODE	Error code. Only one error is defined, and the rest are for user's application.

Bit 2, 1, 0	Error type
000	No error
001	No acknowledge from slave
010	Lines busy or unknown condition

A.3 ODBYTECTR/IDBYTECTR: Output/Input Byte Counter

15		8	7	0
Current Byte Pointer			Number of Bytes	

Bits	Name	Function
15 – 8	Current byte pointer	Points to the current byte. 0 is first byte, 1 is second byte, and so on.
7 – 0	Number of bytes	Number of bytes to send/receive. Since 8-bit, only 256 bytes allow to send/receive at one time

A.4 ODBITCTR/IDBITCTR: Output/Input Bit Counter

15			3	2	0
x					Current Bit Pointer

Bits	Name	Function
15 – 8	–	Not used
7 – 0	Current bit pointer	Points to the current bit poisition of the current byte

A.5 ODPTR/IDPTR: Output/Input Data Pointer

Function: Address of the first memory location of the 256 long data bytes

Appendix B Software Process Flow Chart

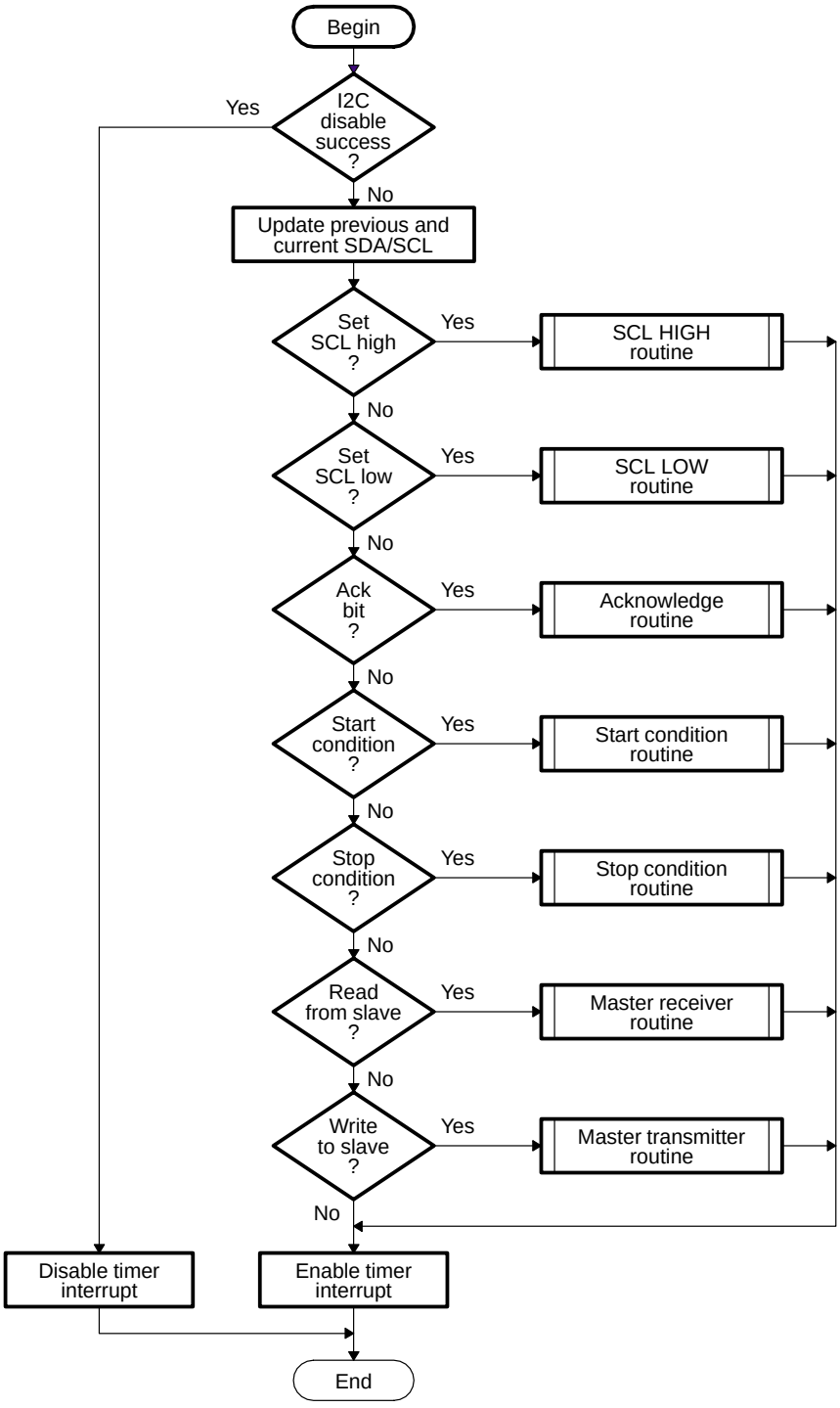


Figure B–1. Software Process Flow Chart of the Implemented I2C Operation

Appendix C Program Listings

```
*******************************************************************************
*The following files are needed to implement the I2C protocol:
*
* 1)  i2c.asm:  timer interrupt service routine
* 2)  vectors.asm:  interrupt vector table
* 3)  init_pll.asm:  initialize CPU clock routine
* 4)  i2c.cmd:  code and data sections allocation
* 5)  i2c.inc:  constants
* 6)  i2creg.inc:  memory registers
* 7)  bsp.inc: McBSP addresses
* 8)  time.inc: timing constants
*******************************************************************************

*******************************************************************************
*   Filename: bsp.inc                                                         *
*******************************************************************************
SPSA         .set 0048H  ; MCBSP1 sub–address register
SPCR1        .set 0049H  ; MCBSP1 control register 1
SPCR2        .set 0049H  ; MCBSP1 control register 2
XCR1         .set 0049H  ; MCBSP1 transmit control register 1
XCR2         .set 0049H  ; MCBSP1 transmit control register 2
SRGR1        .set 0049H  ; MCBSP1 sample rate genarator register 1
SRGR2        .set 0049H  ; MCBSP1 sample rate genarator register 2

SPCR1_SUB    .set 0000H  ; Serial port control register 1 (subaddress)
SPCR2_SUB    .set 0001H  ; Serial port control register 2 (subaddress)
RCR1_SUB     .set 0002H  ; Serial port transmit control register 1 (subaddress)
RCR2_SUB     .set 0003H  ; Serial port transmit control register 1 (subaddress)
XCR1_SUB     .set 0004H  ; Serial port transmit control register 1 (subaddress)
XCR2_SUB     .set 0005H  ; Serial port transmit control register 1 (subaddress)
SRGR1_SUB    .set 0006H  ; Serial port sample rate genarator register 1 (subaddress)
SRGR2_SUB    .set 0007H  ; Serial port sample rate genarator register 2 (subaddress)
PCR_SUB      .set 000EH  ; Serial port pin control register (subaddress)

K_XIOEN      .set 0010000000000000b ; Set XIOEN enable.
K_RIOEN      .set 0001000000000000b ; Set RIOEN enable.
K_FSXM       .set 0000000000000000b ; Set FSX as input.
K_FSRM       .set 0000010000000000b ; Set FSR as output.
K_DX_STAT_1  .set 0000000000100000b ; Mask DX_STAT bit ON.
K_DX_STAT_0  .set 1111111111011111b ; Mask DX_STAT bit OFF.
K_DR_STAT_1  .set 0000000000010000b ; Mask DR_STAT bit ON.
K_DR_STAT_0  .set 1111111111101111b ; Mask DR_STAT bit OFF.
K_FSXP_1     .set 0000000000001000b ; Mask FSXP bit.
K_FSXP_0     .set 1111111111110111b ; Mask FSXP bit.
K_FSRP_1     .set 0000000000000100b ; Mask FSRP bit.
K_FSRP_0     .set 1111111111111011b ; Mask FSRP bit.
```

```
*****************************************************************************
* Filename: i2c.inc                                                         *
* This file defines all constants and flags used in the program.            *
*****************************************************************************

*I2CSTATUS Register
*Status of I2C transmission
*+-------------------------------------------------------------------------------+
*| 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  |  7  | 6  5 |  4 |  3  |  2  |  1  |  0 |
*+-------------------------------------------------------------------------------+
*|DISA|CTRL|SUCC|ERR |ACK |STOP|CLKL|CLKH|Start| Mode |Addr| Pre | Pre | Cur | Cur|
*|    |    |    |    |    |    |    |    |     |      |    | SCL | SDA | SCL | SDA|
*+-------------------------------------------------------------------------------+

K_CSDA_1      .set 0000000000000001b ;Current SDA is high.
K_CSDA_0      .set 1111111111111110b ;Current SDA is low.
K_CSCL_1      .set 0000000000000010b ;Current SCL is high.
K_CSCL_0      .set 1111111111111101b ;Current SCL is low.

K_PSDA_1      .set 0000000000000100b ;Previous SDA is high.
K_PSDA_0      .set 1111111111111011b ;Previous SDA is low.
K_PSCL_1      .set 0000000000001000b ;Previous SCL is high.
K_PSCL_0      .set 1111111111110111b ;Previous SCL is low.

K_ADDR_1      .set 0000000000010000b ;Slave address flag on
K_ADDR_0      .set 1111111111101111b ;Slave address flag off

K_READ_FROM_SLAVE   .set 0000000001000000b ;DSP read from slave
K_WRITE_TO_SLAVE    .set 0000000000100000b ;DSP write to slave
K_READ_FROM_MASTER  .set 0000000000000000b ;DSP read from master
K_WRITE_TO_MASTER   .set 0000000001100000b ;DSP write to master

K_START_1     .set 0000000010000000b ;Start condition
K_START_0     .set 1111111101111111b ;Not start condition

K_CLKH_1      .set 0000000100000000b ;Next interrupt set SCL=high
K_CLKH_0      .set 1111111011111111b ;Disable this feature.

K_CLKL_1      .set 0000001000000000b ;Next interrupt set SCL=low
K_CLKL_0      .set 1111110111111111b ;Disable this feature.

K_STOP_1      .set 0000010000000000b ;Stop condition
K_STOP_0      .set 1111101111111111b ;Disable this feature.

K_ACK_1       .set 0000100000000000b ;Next interrupt is ACK
K_ACK_0       .set 1111011111111111b ;Disable this feature.
```

```
K_ERR_1       .set 0001000000000000b ;Error
K_ERR_0       .set 1110111111111111b ;No Error

K_SUCC_1      .set 0010000000000000b ;I2C data transfer successful
K_SUCC_0      .set 1101111111111111b ;I2C data transfer fails.

K_DISA_1      .set 1000000000000000b ;I2C protocol disable
K_DISA_0      .set 0111111111111111b ;I2C protocol enable

K_RESET_I2C   .set 0                 ;Reset I2C register.

K_I2C_READ    .set 1                 ;Mode of operation: Attach a R/W bit at
K_I2C_WRITE   .set 0                 ;the end of slave address.

*ERRORMSG Register*Error message of I2C
*   +---------------------+
*   | 15     3 |  2   1   0|
*   +---------------------+
*   | xxxx  |   ERRCODE |
*    +----------------------+
K_ERR_NO_ERROR .set 000b
K_ERR_NOACK    .set 001b
K_ERR_UNKNOWN  .set 010b



;*************************
;**USER PERFERENCE SETUP **
;*************************

;***  SETUP SLAVE ADDRESS
*******************************************************************************
K_SLAVEADDR    .set 1010000B     ;Set slave address (7bits).
;*******************************************************************************

;***SELECT SITUATIONS TO STOP DATA TRANSFER*************************************
;K_DISABLE_I2C .set K_SUCC_1     ;Stop after transfer is over.
K_DISABLE_I2C  .set K_SUCC_1|K_ERR_1|K_DISA_1  ;Stop when error or disable occurs.
;*******************************************************************************

***************               EOF        ****************************************
```

```
*******************************************************************************
* Filename: 12creg.inc                                                        *
* This file contains all memory registries in the program.                    *
*******************************************************************************

*********************************************
 VARIABLES
*********************************************
           .align 128
_I2C_DP    .usect “data”,0
_I2CSTATUS .usect “data”,1

*ODBYTECTR
*Output data btye counter
* +--------------------------------+
* |15            8 | 7           0 |
* +--------------------------------+
* | Current byte    |   Number of  |
* | pointer         |     byte     |
* +--------------------------------+
*   byte pointer : 8bits . because reserved 255 words
*
*ODBITCTR
*Output data bit counter
* +--------------------------------+
* |15                  4 | 3     0 |
* +--------------------------------+
* |        XXXXX         |   Bit   |
* |                      | pointer |
* +--------------------------------+
*   bit pointer : 3bits . for byte

_ODBYTECTR  .usect “data”, 1
ODBITCTR    .usect “data”, 1

*IDCTR
*Input data btye/bit counter
* +--------------------------------+
* |15            8 | 7           0 |
* +--------------------------------+
* | Current byte    |   Number of  |
* | pointer         |     byte     |
* +--------------------------------+
*   byte pointer : 8bits . because reserved 255 words
*
```

```
*IDBITCTR
*Input data bIT counter
* +--------------------------------+
* |15                  4 | 3     0 |
* +--------------------------------+
* |        XXXXX         |   Bit   |
* |                      | pointer |
* +--------------------------------+
*   bit pointer : 3 bits for a byte

_IDBYTECTR   .usect “data”,1
IDBITCTR     .usect “data”,1
ERRORCODE    .usect “data”,1
_SLAVE_ADDR  .usect “data”,1
TEMP_REG     .usect “data”,1
POINTER      .usect “data”,1
TEMP1_REG    .usect “data”,1

*ODPTR
*Output data address pointer register (16bits)
*reserved 255 words (uses only 8 bits)
* +--------------------------------+
* |15                            0 |
* +--------------------------------+
* | Address of output data         |
* +--------------------------------+

             .align 128
_ODPTR       .usect “data”, 0ffh

*IDPTR
*Input data address pointer register
*reserved 255 words (uses only 8 bits)
* +--------------------------------+
* |15                            0 |
* +--------------------------------+
* | Address of input data          |
* +--------------------------------+
             .align 128
IDPTR        .usect “data”, 0ffh

**************************END OF FILE***********************************
```

```
*****************************************************************
* Filename: time.inc                                            *
* This file contains the programmable period of timer interrupts. *
*                                                               *
*****************************************************************

;***USER PERFERENCE SETUP**************************************
F_CPU_MHz          .set 80 ;If init_pll.asm is used, CPU
                           ;Frequency is 80MHz.
PRD_VAL_us         .set 1  ;Select to obtain 1us interrupt.
START_HOLD_TIME_u  .set5   ;Select to obtain 5us interrupt.
SCL_HIGH_us        .set5   ;Select to obtain 5us interrupt.
SCL_LOW1_us        .set5   ;Select to obtain 5us interrupt.
SCL_LOW2_us        .set5   ;Select to obtain 5us interrupt.
STOP_SETUP_TIME_us .set5   ;Select to obtain 5us interrupt
;**END OF USER PERFERENCE SETUP********************************

K_PRD_VAL .set ((PRD_VAL_us*F_CPU_MHz)/16)–1
K_START_HOLD_TIME  .set  ((START_HOLD_TIME_us*F_CPU_MHz)/16)–1
K_SCL_HIGH         .set  ((SCL_HIGH_us*F_CPU_MHz)/16)–1
K_SCL_LOW1         .set  ((SCL_LOW1_us*F_CPU_MHz)/16)–1
K_SCL_LOW2         .set  ((SCL_LOW2_us*F_CPU_MHz)/16)–1
K_STOP_SETUP_TIME  .set  ((STOP_SETUP_TIME_us*F_CPU_MHz)/16)–1
STOP_TIMER         .set  001FH
START_TIMER        .set  002FH

**************************END OF FILE*******************************
```

```
/**********************/
/*                    */
/* Filename : i2c.cmd */
/* –define memory map */
/*                    */
/**********************/

–e start   /*define starting point of program*/
MEMORY
{
  PAGE 0:
     P_VECT    (RWX)  : org = 0080h, len = 0080h
     P_DARAM   (RWX)  : org = 2000h, len = 100h

  PAGE 1:
     D_SARAM2  (RWX)  : org = 2100h, len = 700h
}
SECTIONS
{
     vectors          : {} > P_VECT   PAGE 0
     .text            : {} > P_DARAM  PAGE 0
     data             : {} > D_SARAM PAGE 1
     constant         : {} > D_SARAM2 PAGE 1
     program          : {} > D_SARAM2 PAGE 1
     STK              : {} > D_SARAM2 PAGE 1
}

/************************END OF FILE***********************************/
```

```
/************************/
/*                      */
/* Filename: init.gel   */
/* –the GEL file for CCS */
/*                      */
/************************/

/* set PMST to */
/* IPTR = 01H */
/* MP = OVLY = 1; DROM off, CLKOUT on */
#define PMST            0x1d
#define PMST_VAL        0x00e0

/* set wait–state control reg for: 2 w/s for i/o, ext/int data memory */
#define SWWSR           0x28
#define SWWSR_VAL       0x2492


/* set external–banks switch control for: no bank switching; BH set */
#define BSCR            0x29
#define BSCR_VAL        0x02

StartUp()
{
 init();
}
menuitem “C5400”;

hotmenu reset()
{
        GEL_Reset();
        init();
        GEL_MapOn();
        GEL_MapReset();
        GEL_XMDef(0,0x1e,1,0x8000,0x7f);
        GEL_XMOn();

        GEL_MapAdd(0,1,0x8000,1,1);
        GEL_MapAdd(0,0,0x20000,1,1);
 GEL_MapAdd(0,2,0xffff,1,1);
}
hotmenu init()
{
        *(int *)PMST = PMST_VAL;
        *(int *)SWWSR = SWWSR_VAL;
        *(int *)BSCR = BSCR_VAL;
        GEL_MemoryFill(0x4,2,1,0xff03);
}
```

```
*****************************************************************
*                                                               *
* Filename: init_pll.asm                                        *
* This file initializes the CPU clock (10-MHz crystal is used). *
*                                                               *
*****************************************************************
       .mmregs
       .def       _INIT_PLL
       .text

_INIT_PLL:

;-- 1. Set clock mode to DIV mode. In order to change PLLNDIV, PLLCOUNT etc
       STM #0,CLKMD.
wait_div_mode:
       BITF *(CLKMD),#1       ;Test if PLLSTSATUS=0.
       BC wait_div_mode,TC    ;If not, move on(PLL mode).

;-- 2. Set clock mode to PLL mode.
       STM #0111011111111111b,CLKMD
           ;0111~~~~~~~~~~~~  ;PLL multiplier, freq = PLLMUL + 1.
           ;~~~~0~~~~~~~~~~~  ;PLLDIV, pll divide mode
           ;~~~~~11111111~~~  ;PLLCOUNT, pll lock time, I/P clk X 16
           ;~~~~~~~~~~~~~1~~  ;PLLONOFF
           ;~~~~~~~~~~~~~~1~  ;PLLNDIV, pll divide mode
           ;~~~~~~~~~~~~~~~1  ;PLLSTATUS, pll mode(read only)

;-- 3. Wait for PLL lock,
wait_pll_mode:
       BITF *(CLKMD),#1
       BC wait_pll_mode,NTC
       NOP

       LDM  PMST,A
       OR  #4H,A              ;Enable CLKOUT signal.
      STLM A,PMST
      NOP
      NOP
      RET

*****************************END OF FILE*********************************
```

```
*************************************************************************
* Filename: main.asm                                                    *
* This is an example to test the I2C data transfer using EEPROM PCF8582. Its  *
* address is defined in i2c.asm K_SLAVEADDR.                            *
*                                                                       *
*************************************************************************

     .mmregs
     .def start
     .ref _INIT_PLL,_ODBYTECTR,_IDBYTECTR,_ODPTR
     .ref _init_gpio,_init_i2c,_write_i2c,_read_i2c
     .def _USER_FUNCTION


LEN  .set  400    ;Length of stack segment
BOS  .usect  "STK",LEN

;************************************************************************
; MAIN ROUTINE
;************************************************************************
     .text
start:
     STM   #BOS+LEN,SP     ;Setup stack pointer.
     CALL  _INIT_PLL       ;Init CPU clock.
     RSBX  SXM             ;Sign extension suppressed
     NOP
     CALL  _init_gpio      ;Init McBSP to GPIO.
     CALL  _init_i2c       ;Init I2C data transfer.
;************************************************************************

;********SET UP SLAVE ADDR AND WRITE DATA (DATA MUST BE BYTE!!!)**********
     CALL  _init_i2c        ;Init I2C data transfer.
     STM   #_ODPTR+1,AR2    ;_ODPTR+0 contains slave addr.
     ST    #0,*AR2+         ;Initiate EEPROM internal addr pointer 00h.
     ST    #011H,*AR2+      ;Store 11h to EEPROM addr 00h.
     ST    #012H,*AR2+      ;Store 12h to EEPROM addr 01h.
     ST    #013H,*AR2+      ;Store 13h to EEPROM addr 02h.
     ST    #014H,*AR2+      ;Store 14h to EEPROM addr 03h.
     ST    #015H,*AR2+      ;Store 15h to EEPROM addr 04h.
     ST    #016H,*AR2+      ;Store 16h to EEPROM addr 05h.
     STM   #_ODBYTECTR,AR1
     ST    #7,*AR1          ;7 bytes to be sent, including slave addr.
     CALL   _write_i2c      ;wWite and enable global interrupt.

     CALL Delay             ;Allows for E/W cycle of EEPROM.
```

```
;*********SET UP EEPROM Internal Address Pointer*************************
     CALL   _init_i2c       ;Init I2C data transfer.
     STM    #_ODPTR+1,AR1   ;_ODPTR+0 contains slave addr.
     ST     #0,*AR1+        ;Reset EEPROM internal addr pointer back to 00h.
     NOP
     STM    #_ODBYTECTR,AR1
     ST     #1,*AR1         ;1 byte to be sent.
     CALL   _write_i2c      ;Write and enable global interrupt.

     CALL   Delay           ;Allows for E/W cycle of EEPROM


;*********READ BACK 6 BYTES FROM EEPROM**************************************
     CALL   _init_i2c       ;Init I2C data transfer.
     STM    #_IDBYTECTR,AR1 ;1ST loc at odptr contains slave addr.
     ST     #6,*AR1         ;Read in 6 bytes.
     CALL   _read_i2c       ;Read and enable global interrupt.
;*********End of program*****************************************************
STOPSTOP:
     B STOPSTOP

;***********************************************************************
;***OTHER SUBROUTINES*****************************************************
;***********************************************************************
Delay:           ;A delay subroutine to allow for the E/W cycle  NOP
                 ;before any further write/read process to EEPROM
       PSHM  AR5
       PSHM  AR6
       NOP
       STM   #2000,AR5
LOOP1: STM   #2000,AR6
LOOP2: NOP
       NOP
       BANZ  LOOP2,*AR6-
       BANZ  LOOP1,*AR5-
       NOP
       POPM  AR6
       POPM  AR5
       NOP
       RET
;***********************************************************************
;***********************************************************************
_USER_FUNCTION:     ;User function subroutine is called when data
       NOP          ;transfer is successful.
       RET

.end
```

```
*****************************END OF FILE*********************************
*************************************************************************
*                                                                       *
* Filename: i2c.asm                                                     *
* This files the I2C timer interrupt service routine and other I2C      *
* initialization routines.                                              *
*                                                                       *
*************************************************************************
       .mmregs
       .include “time.inc”
       .include “i2c.inc”
       .include “bsp.inc”
       .include “i2creg.inc”

       .def TIMER_INT
       .def _I2CSTATUS,_ODBYTECTR,_IDBYTECTR,_ODPTR,_SLAVE_ADDR
       .def _init_gpio,_init_i2c,_write_i2c,_read_i2c
       .ref _USER_FUNCTION

;***MASK array defines the bit position of a byte
    .sect “constant”
MASK   .int 01h  ;Bit 0
       .int 02h  ;Bit 1
       .int 04h  ;Bit 2
       .int 08h  ;Bit 3
       .int 10h  ;Bit 4
       .int 20h  ;Bit 5
       .int 40h  ;Bit 6
       .int 80h  ;Bit 7 (MSB)

       .sect “program”
;************************************************************************
; TIMER ISR
;************************************************************************
TIMER_INT:

;***SAVE CPU REGISTERS TO STACK***
       PSHM ST0
       PSHM ST1
       PSHM AR1
       PSHM AR2
       PSHM AR3
       PSHM AR4
```

```
       PSHM AR5
       PSHM AR6
       NOP
       NOP
       RSBX   SXM                      ;Suppressed sign extension
       RSBX   CPL                      ;Use DP as relative direct–addressing.
       RSBX   HM                       ;Hold  mode = 0.
       LD     #_I2C_DP,DP              ;Set DP.
       NOP
       NOP
       LD _   I2CSTATUS,A              ;Load i2c status.
       LD     A,B                      ;Duplicate i2c status.
       AND    #K_DISABLE_I2C,B         ;Check if i2c disable.
       BC     tint_disable_tint,BNEQ   ;If =1, go to end.
;***UPDATE PREVIOUS SDA AND SCL***
       AND    #3H,A                    ;Mask all except current bits.
       LD     A,2,B                    ;Shift current bits to previous bits.
       ANDM   #0FFF0H,_I2CSTATUS       ;Mask off lower 4 bits.
       LD     _I2CSTATUS,A             ;Load i2c status.
       ADD    B,A                      ;Current –> previous
       STL    A,_I2CSTATUS             ;Store back to register (current=0).
;***UPDATE CURRENT SDA AND SCL***
       STM    #SPSA,AR2                ;AR2 points to SPSA.
       ST     #PCR_SUB,*AR2+           ;Check line status of SCL.
       LD     #(K_DR_STAT_1|K_FSXP_1),B  ;Mask DR_STAT & FSXP bit (bits 4 and 3).
       AND    #0FFFFH,A                ;Clear AH(i2c status).
       AND    *AR2–,B                  ;Check DR_STAT and FSXP bit.
       OR     B,–3,A                   ;Shift current SCL to bits 1 and 0.
       STL    A,_I2CSTATUS             ;store back to i2c status register.
;***CHECK START/STOP CONDITION OR MODE OF OPERATION***
       LD     A,B                      ;Duplicate i2cstatus.
       AND    #K_CLKH_1,B              ;Need to set SCL=high.
       BC     tint_clk_go_high,BNEQ    ;If one, goto set_high
       LD     A,B                      ;Duplicate i2c status.
       AND    #K_CLKL_1,B              ;Need to set SCL=low.
       BC     tint_clk_go_low,BNEQ     ;If one, goto set_low.
       LD     A,B                      ;Duplicate i2c status.
       AND    #K_ACK_1,B               ;Check ACK in i2c status.
       BC      tint_ack,BNEQ           ;If ACK=1, goto tinit_ack.
       LD      A,B                     ;Duplicate i2cstatus.
       AND    #K_START_1,B             ;Check start condition in i2c status.
       BC    tint_start,BNEQ           ;If start=1, goto tinit_start.
```

```
       AND   #K_STOP_1,A                ;Check stop condition in i2c status.
       BC    tint_stop,ANEQ             ;If stop=1, go to tinit_stop.
       MVDK  _I2CSTATUS,TEMP_REG        ;Copy i2c status to temp register.
       ANDM  #01100000B, TEMP_REG       ;Mask on 6 and 5 bits (Mode of operation).
       CMPM  TEMP_REG,#K_READ_FROM_SLAVE ;Compare only 6 and 5 bits.
       BC    tint_read_slave, TC        ;If 10, go to read_slave(TEMP_REG as an
                                        ;extra ACK flag, not equal to 0).
       CMPM  TEMP_REG,#K_WRITE_TO_SLAVE  ;Compare only 6 and 5 bits.
       BC    tint_write_slave, TC       ;If 01,goto write slave(TEMP_REG
                                        ;as a extra ACK flag, not equal to 0).
                                        ; ||
       B     tint_busy                  ;For future upgrades,
                                        ;multi–master not supported now
 ;***END OF MAIN CONTROL LOOP***
;************************************************************************
;************************************************************************

;*****START CONDITION SEQUENCES*****
tint_start:
;***READ THE PREVIOUS AND CURRENT SDA/SCL LEVELS***
       MVDK  _I2CSTATUS,TEMP_REG        ;Copy i2c status to temp register.
       ANDM  #0FH, TEMP_REG             ;Mask on lower 4 bits.
       CMPM  TEMP_REG,#1111B            ;All lines high
       BC    tint_start_start,TC        ;Go to end of start condition.
       CMPM  TEMP_REG,#1110B            ;Start finished
                ;1~~~                   ;Previous SCL=1.
                ;~1~~                  ;Previous SDA=1.
                ;~~1~                  ;Current SCL=1.
                ;~~~0                  ;Current SDA=0.
       BC    tint_start_finish,TC       ;Go to end of start condition.
       B     tint_start_invalid         ;Go to invalid (only consider 2 cases).
;***BEGIN OF START CONDITION***
tint_start_start:
       CALL  pullSDA                    ;SDA–>0 while SCL=1.
       ;***SETUP TIMER INTERRUPT FOR hold time=5us
       STM   #STOP_TIMER,TCR            ;Stop timer.
       STM   #K_START_HOLD_TIME,PRD     ;Load hold time of CLK (5us).
       STM   #START_TIMER,TCR           ;Start timer.
       B     tint_end_isr
;***END OF START CONDITION***
tint_start_finish:
       CALL  pullSCL                    ;SCL=0 after hold time.
```

```
       ANDM  #K_START_0,_I2CSTATUS       ;Clear start flag.
       ST    #1,TEMP_REG                 ;Set extra ACK flag for normal clocking.
       B     tint_end_isr                ;TINT still at 5us
;***UNKNOWN STATE***
tint_start_invalid:
       CALL  releaseSDA                  ;Release both lines and do start again.
       CALL  releaseSCL                  ;
       B     tint_end_isr                ;End timer ISR.

;**************************************************************************
;**************************************************************************

;******BEGIN ACKNOWLEDGEMENT BIT SEQUENCE*****
tint_ack:
       ;***CHECK IF MASTER-RECEIVER MODE (read slave)
       MVDK  _I2CSTATUS,TEMP_REG        ;Copy i2c status to temp register.
       ANDM  #01100000B, TEMP_REG       ;Mask on 6 and 5 bits (Mode of operation).
       CMPM  TEMP_REG,#K_READ_FROM_SLAVE ;Compare only 6 and 5 bits.
       BC    tint_ack_not_rs, NTC        ;If not 10,go to not_rs.
       LD    I2CSTATUS,A                 ;Check if sending of slave addr is over?
       AND   #K_ADDR_1,A                 ;
       BC    tint_ack_end_slave_addr,ANEQ ;If end of send addr, go to end_slave_addr.
       LD    POINTER,B                   ;Extra flag
       BC    tint_ack_rs_first,BEQ       ;If first time, POINTER==0.
       CALL  releaseSDA
       ANDM  #K_ACK_0,_I2CSTATUS         ;Remove ACK flag.
       B     tint_end_isr
tint_ack_rs_first:
       ST    #1,POINTER                  ;Set POINTER flag.
       LD    _I2CSTATUS,A                ;Load i2c status.
       AND   #K_STOP_1,A                 ;Check stop condition in i2c status.
       BC    tint_ack_stop,ANEQ          ;If stop=1, go to tint_ack_stop.
       CALL  pullSDA                     ;ACK is LOW when not last byte
B tint_ack_end                           ;TEMP_REG flag is not 0.
tint_ack_stop:
       CALL  releaseSDA ;ACK is HIGH when last byte
       B     tint_ack_end                ;TEMP_REG flag is not 0.

tint_ack_end_slave_addr:
       ANDM  #K_ADDR_0,_I2CSTATUS        ;Remove slave address flag.
                                         ;Continue on, let slave ACK.
```

```
;***IN MASTER–TRANSMIT MODE (write slave)***
tint_ack_not_rs:
       CALL  releaseSDA                ;SDA=high. Let slave ACK on this line.
       ST    #0,TEMP_REG               ;Clear temp reg, as an extra ACK flag.
tint_ack_end:
       ORM   #K_CLKH_1,_I2CSTATUS      ;Next TINT, enable SCL –> high.
       B     tint_end_isr

;***TEST IF SLAVE ACKNOWLEDGE TO MASTER***
tint_ack_test:                         ;Call from where SCL=high(CLKH flag removed).
       ;***CHECK IF SLAVE ACK BY PULLING SDA DOWN***
       ST    #PCR_SUB,*AR2+            ;Sub-bank address (SPSA)=PCR.
       LD    #K_FSXP_1,A               ;Check if SDA go low?
       AND   *AR2-,A
       BC    tint_ack_sda_low,AEQ      ;If SDA=low, go to sda_low.
       ;***SLAVE DID NOT ACKNOWLEDGE***
       ORM   #K_ERR_1,_I2CSTATUS       ;Turn error flag on.
       ST    #K_ERR_NOACK,ERRORCODE    ;Give type of error.
tint_ack_sda_low:
      ;***SLAVE ACKNOWLEDGED***        ;SCL go_high flag has disabled.
      ORM   #K_CLKL_1,_I2CSTATUS       ;Enable SCL go_low flag.
      ANDM  #(K_ACK_0&K_ADDR_0),_I2CSTATUS  ;Remove ACK and ADDR flag.
      STM   #STOP_TIMER,TCR            ;Stop timer.
      STM   #K_SCL_HIGH,PRD            ;Load high period of SCL (at 5us).
      STM   #START_TIMER,TCR           ;Start timer.
      B     tint_end_isr               ;Interrupt at 5us.

;************************************************************************
;************************************************************************

;*****STOP CONDITION SEQUENCES*****
tint_stop:                             ;SCL=low after ACK.
       ;***CHECK IF FIRST TIME ENTER STOP SEQUENCE***
      LD    TEMP1_REG,A                ;To make sure transition of SDA does
      BC    tint_stop_continue,ANEQ    ;not occur in SCL=1
      CALL  pullSDA                    ;If extra ACT flag=1, not first time
      ST    #1,TEMP1_REG               ;Set flag=1.
      STM   #STOP_TIMER,TCR            ;Stop timer.
      STM   #K_STOP_SETUP_TIME,PRD     ;Load stop setup time (5us).
      STM   #START_TIMER,TCR           ;Start timer.
      B     tint_end_isr               ;Check again.
```

```
tint_stop_continue:
      MVDK  _I2CSTATUS,TEMP_REG         ;Copy i2c status to temp register.
      ANDM  #03H,TEMP_REG               ;Mask on lower 2 bits.
      ;***AFTER 2ND PASS, SCL=1 , SDA=0***
      CMPM  TEMP_REG,#10B               ;Stop finished
                  ;1~                   ;Current SCL=1.
                  ;~0                   ;Current SDA=0.
      BC    tint_stop_finish,TC         ;Go to end of stop condition.
      ;***IMMEDIATELY AFTER ACK, SCL/SDA=0/0***
      CMPM  TEMP_REG,#00B               ;After ACK
                  ;0~                   ;Current SCL=0.
                  ;~0                   ;Current SDA=0.
      BC    tint_stop_invalid,NTC       ;Go to invalid (only consider 2 cases).
      ;***CHECK IF SLAVE HOLD ON TO SCL LINE***
      CALL  releaseSCL                  ;SCL=1.
      NOP                               ;Wait for a while. (Does SCL go high
      NOP                               ;Wait for a while.   immediately?)
      NOP                               ;Wait for a while.
      ;***SYNCHRONIZATION***
      ST    #PCR_SUB,*AR2+              ;Sub-bank address (SPSA)=PCRLD #K_DR_STAT_1,A.
                                        ;Check if SCL go high?
      AND   *AR2-,A
      BC    tint_stop_scl_go_high_yes,ANEQ  ;If SCL=high, goto high_yes.
      ;***SLAVE STILL HOLDING ON TO SCL LINE***
      ;***SETUP TIMER INTERRUPT FOR 1us***
      B     tint_stop_invalid           ;Check again.


tint_stop_scl_go_high_yes:
      ;***SETUP TIMER INTERRUPT FOR hold time=5us***
      STM   #STOP_TIMER,TCR             ;Stop timer.
      STM   #K_STOP_SETUP_TIME,PRD      ;Load stop setup time (5us).
      STM   #START_TIMER,TCR            ;Start timer.
      B     tint_end_isr
      ;***SDA=0 -> 1***
tint_stop_finish:
      CALL  releaseSDA                  ;SDA=1.
      ANDM  #K_STOP_0,_I2CSTATUS        ;Clear stop flag.
      ORM   #K_SUCC_1,_I2CSTATUS        ;SUCCESSFUL
      NOP
     CALL   _USER_FUNCTION              ;Go to user-function subroutine.
     NOP
     B      tint_end_isr               ;End timer ISR.
```

```
tint_stop_invalid:
     CALL   pullSDA                    ;Enable the situation of stop condition.
     CALL   releaseSCL                 ;=> both lines are low.
     STM    #STOP_TIMER,TCR            ;Stop timer.
     STM    #K_PRD_VAL,PRD             ;Load PRD (1us).
     STM    #START_TIMER,TCR           ;Start timer.
     B      tint_end_isr               ;Check again.

;***************************************************************************
;***************************************************************************

;*****READ FROM SLAVE SEQUENCES*****
tint_read_slave:
     LD    _I2CSTATUS,A              ;Load i2c status.
     AND    #K_ADDR_1,A              ;Check if send slave addr now.
     BC     tint_clk_go_high,AEQ     ;If =0, go to rs_mode.
     B      tint_write_slave         ;Write slave addr in the first byte.
;***READ DATA FROM SLAVE DURING SCL=HIGH***
tint_rs_mode:
    ;***SETUP POINTERS TO CORRECT LOCATION***
     LD     #IDPTR,B                 ;Load addr of out–data to B.
     LD     _IDBYTECTR, –8,A         ;Move current byte pointer.
     STL    A,POINTER                ;Store it to POINTER (1ST = BYTE 0).
     ADD    A,B                      ;Update current btye pointer.
     STLM   B,AR4                    ;AR4 points to current byte.

     CMPM   IDBITCTR,#7              ;If IDBITCTR == 7, then it is new byte,
     BC     tint_rs_not_new_byte,NTC ;else not new byte.
     ST     #0,*AR4                  ;Clear the memory location.
tint_rs_not_new_byte:
     ST     #PCR_SUB,*AR2+           ;Check current line status of SDA.
     LD     #K_FSXP_1,B              ;Mask FSXP bit (bit 3).
     AND    *AR2–,B                  ;Check FSXP bit.
     LD     B,–3,A                   ;Store to bit 0 of A.

     LD     IDBITCTR,B               ;Check if end of byte,i.e., bit ctr=0.
     BC     tint_rs_bit_0,BEQ        ;If bit ctr=0, go to new_byte,
     SUB    #1,B                     ;else decrement bit ctr.
     STL    B,TEMP1_REG              ;Store to TEMP1 register.
```

```
     RPT    TEMP1_REG               ;Shift A (bitctr–1) times.
     SFTL   A,1                     ;Result in A.
    ;***LSB NO NEED TO SHIFT***
tint_rs_bit_0:                      ;If bit 0, no need to shift.
     LD     *AR4,B                  ;Load B with current read byte.
     OR     A,B                     ;Combine and store to B.
     AND #0FFH,B                    ;Allows only byte
     STL B,*AR4                     ;Store back to memory.

     LD IDBITCTR,A                  ;If BITCTR=0(LSB) ie., end of byte,
     BC tint_rs_next_byte,AEQ       ;go to next byte.
     SUB #1,A,B                     ;Decrement A and store to B.
     STL B,IDBITCTR                 ;Store back to register.
     B tint_rs_next                 ;Next bit
     ;***TEST IF COME TO LAST BYTE***
tint_rs_next_byte:
     ORM #K_ACK_1,_I2CSTATUS        ;Enable ACK flag.
     ST #7,IDBITCTR                 ;Start from MSB (initiate next byte).
     ADDM #1,POINTER                ;Increment POINTER.
     ANDM #0FFH,_IDBYTECTR          ;Prepare num of btye to receive.

     LD POINTER,B                   ;Load the incremental POINTER to B.
     LD _IDBYTECTR,A                ;Load num of byte to A.
     SUB B,A                        ;A=(Num of byte) – (current byte POINTER).
     BC tint_rs_no_more_byte,AEQ    ;If=0, no more to receive.
     LD POINTER,8,A                 ;Shift 8 places forward.
     LD _IDBYTECTR,B                ;
     OR B,A                         ;Combine POINTER and num of btye to send.
     STL A,_IDBYTECTR               ;Update IDBYTECTR.
     B tint_rs_next                 ;Go to next byte.
tint_rs_no_more_byte:
     ORM #K_STOP_1,_I2CSTATUS       ;Enable STOP condition.
tint_rs_next:
     ST #0,POINTER                  ;Extra flag use in ack_rs.
     ORM #K_CLKL_1,_I2CSTATUS       ;Next TINT, enable releaseSCL.
     STM #STOP_TIMER,TCR            ;stop timer.
     STM #K_SCL_HIGH,PRD            ;Load PRD (interrupt at 5us).
     STM #START_TIMER,TCR           ;Start timer.
     B tint_end_isr ;

;***************************************************************************
;***************************************************************************
```

```
;*****WRITE TO SLAVE SEQUENCES*****
tint_write_slave:
     ;***SET UP AR4 POINTS TO BYTE TO BE SENT***
     LD #_ODPTR,B                    ;Load addr of out–data to B.
     LD _ODBYTECTR, –8,A             ;Move current byte pointer.
     STL A,POINTER                   ;Store it to POINTER (1ST= BYTE 0).
     ADD A,B                         ;Update current btye pointer.
     STLM B,AR4                      ;AR4 points to current byte.
     ;***SET UP AR3 POINTS TO BIT TO BE SENT***
     STM #MASK,AR3                   ;AR3 points to MASK.
     RPT ODBITCTR                    ;Look up table (LUT).
     LD *AR3+,A                      ;Store mask to A.
     AND *AR4, A                     ;Check if the bit is high or low.
     ;***WRITE BIT***
     CC releaseSDA,ANEQ              ;Bit=1, SDA=high.
     CC pullSDA,AEQ                  ;Bit=0, SDA=low.
     ;***TEST IF COME TO LAST BIT***
     LD ODBITCTR,A                   ;If BITCTR=0(LSB) i.e., end of byte,
     BC tint_ws_next_byte,AEQ        ;go to next byte.
     SUB #1,A,B                      ;Decrement A and store to B.
     STL B,ODBITCTR                  ;Store back to register.
     B tint_ws_next                  ;Next bit
     ;***TEST IF COME TO LAST BYTE***
tint_ws_next_byte:
     LD POINTER,B                    ;First byte is byte zero.
     ORM #K_ACK_1,_I2CSTATUS         ;Enable ACK flag.
     ST #7,ODBITCTR                  ;Start from MSB (initiate next byte).
     ADD #1,B                        ;Increment POINTER.
     STL B,POINTER                   ;Store back.
     ANDM #0FFH,_ODBYTECTR           ;Prepare num of btye to send.
     LD _ODBYTECTR,A                 ;Load num of byte to A.
     SUB B,A ;A=(Num of byte) – (current byte POINTER)
     BC tint_ws_end_write,AEQ        ;If=0, no more to send. Enable stop condition.
     LD POINTER,8,A                  ;Shift 8 places forward.
     LD _ODBYTECTR,B                 ;
     OR B,A                          ;Combine POINTER and num of btye to send.
     STL A,_ODBYTECTR                ;Update ODBYTECTR.
     B tint_ws_next                  ;Go to next byte.
     ;***NO MORE DATA***
tint_ws_end_write:                   ;Do not update POINTER for last byte.
     ORM #K_STOP_1,_I2CSTATUS        ;Enable STOP condition.
```

```
     ST #0,TEMP1_REG
     ;***PREPARE FOR NEXT BIT***
     tint_ws_next:
     ORM #K_CLKH_1,_I2CSTATUS         ;Next TINT,e nable releaseSCL.
     STM #STOP_TIMER,TCR              ;Stop timer.
     STM #K_SCL_LOW1,PRD              ;Load SCL low period(interrupt at 5us).
     STM #START_TIMER,TCR             ;Start timer.
     B tint_end_isr ;
     ;***BUSY LINES OR UNKNOWN CONDITION
tint_busy:                            ;Unknown condition or lines are busy.
     ;***REPORT ERROR TYPE***
     ORM #K_ERR_1,_I2CSTATUS          ;Turn error flag on.
     ST #K_ERR_UNKNOWN,ERRORCODE      ;Give type of error.
     B tint_end_isr
;**************************************************************************
;**************************************************************************
;*****MAKE CLK->HIGH SEQUENCES*****
tint_clk_go_high:
     CALL releaseSCL                  ;SCL=high.
     NOP                              ;Wait for a while. (Does SCL go high
     NOP                              ;Wait for a while.    immediately?)
     NOP                              ;Wait for a while.
      ;***SYNCHRONIZATION***
     ST #PCR_SUB,*AR2+                ;Sub-bank address (SPSA)=PCR.
     LD #K_DR_STAT_1,A                ;Check if SCL go high?
     AND *AR2-,A
     BC tint_clk_go_high_yes,ANEQ     ;If SCL=high, goto high_yes.
     ;***SLAVE HOLD SCL LOW***
     STM #STOP_TIMER,TCR              ;Stop timer.
     STM #K_PRD_VAL,PRD               ;Load PRD (interrupt at 1us).
     STM #START_TIMER,TCR             ;Start timer.
     B tint_end_isr                   ;Do go_high procedure again.
;***CLK GONE HIGH (SLAVE NO MORE HOLDING ONTO SCL)***
tint_clk_go_high_yes:
     ANDM #K_CLKH_0,_I2CSTATUS        ;Remove SCL go_high flag.
;*****ACKNOWLEDGE SEQUENCE***
     ;***CHECK IF IT IS IN THE ACK PHASE (write slave)***
     LD TEMP_REG,A                    ;Check if Slave release SDA line.
     BC tint_ack_test,AEQ             ;Extra ACT flag=0: ACK in progress
      ;*****END OF ACKNOWLEDGE SEQUENCE***
      ;*****READ THE CURRENT SDA LEVEL (FOR READ SLAVE MODE)***
      ;***CHECK IF MASTER-RECEIVER MODE (read slave)***
```

```
      LD _I2CSTATUS,B                 ;
      LD B,A ;Duplicate
      AND #K_ADDR_1,A                 ;Check ADDR in i2c status.
      BC tint_clk_go_high_not_rs,ANEQ  ;If sending slave addr, go to not_rs.

      MVDK _I2CSTATUS,TEMP_REG        ;Copy i2c status to temp register.
      ANDM #01100000B, TEMP_REG       ;Mask on 6 and 5 bits (mode of operation).
      CMPM TEMP_REG,#K_READ_FROM_SLAVE ;Compare only 6 and 5 bits.
      BC tint_clk_go_high_not_rs,NTC  ;If not 10, go to not_rs.
      ;***CHECK IF IN ACK PHASE***
      AND #K_ACK_1,B                  ;Check ACK in i2c status.
      BC tint_rs_mode,BEQ             ;If ACK=0, goto rs_mode.
                                      ;***CHECK IF GOING TO STOP SEQUENCE***
      LD _I2CSTATUS,A                 ;Check if come to the last byte
      AND #K_STOP_1,A                 ;Need to set extra TEMP_REG flag=0
      BC tint_clk_go_high_not_rs,AEQ  ;Bcos to avoid the unknown transition
                                      ;of SDA.
      ST #0,TEMP1_REG                 ;To prolong SDA=0 in stop sequences
;*****END OF READ THE CURRENT SDA LEVEL (FOR READ SLAVE MODE)***
tint_clk_go_high_not_rs:
      ;***END OF READ SLAVE MODE***
      ORM #K_CLKL_1,_I2CSTATUS        ;Enable SCL go_low flag.
      STM #STOP_TIMER,TCR             ;Stop timer.
      STM #K_SCL_HIGH,PRD             ;Load PRD (interrupt at 5us).
      STM #START_TIMER,TCR            ;Start timer.
      B tint_end_isr                  ;Interrupt at 5us.
;*************************************************************************
;*************************************************************************
;***MAKE CLK->LOW SEQUENCES***
tint_clk_go_low:
      CALL pullSCL                    ;SCL=low.
      ANDM #K_CLKL_0,_I2CSTATUS       ;Remove SCL go_low flag.
      STM #STOP_TIMER,TCR             ;Stop timer.
      STM #K_SCL_LOW2,PRD             ;Load SCL low period(interrupt at 5us).
      STM #START_TIMER,TCR            ;Start timer.
      B tint_end_isr                  ;Interrupt at 5us.


;*************************************************************************
;*************************************************************************


;***DISABLE FURTHER TIMER INTERRUPT***
tint_disable_tint:
```

```
      ANDM #0FFF7H,IMR                ;Disable any further TINT.
      B    tint_end_end

;******END OF ISR*****
tint_end_isr:
     ORM #0008H,IMR ;enable timer interrupt
     STM #0FFFFH,IFR                  ;Clear any pending interrupts.
;***START OF OTHER TINT ISR
tint_end_end:
     ;***RESTORE CPU REGISTERS
     NOP
     NOP
     POPM AR6
     POPM AR5
     POPM AR4
     POPM AR3
     POPM AR2
     POPM AR1
     POPM ST1
     POPM ST0
     NOP
     NOP
                                      ;END OF TIMER ISR.
     RETE                             ;Enable global INTM.

;*************************************************************************
;*************************************************************************


;***
;OTHER SUBROUTINES
;***



;*************************************************************************
; SDA = HIGH ISR
;*************************************************************************
releaseSDA:
     ST #PCR_SUB,*AR2+                ;Sub-bank address (SPSA0)=PCR0.
     ORM #K_DX_STAT_1, *AR2-          ;Mask on DX_STAT BIT.
     NOP
     NOP
```

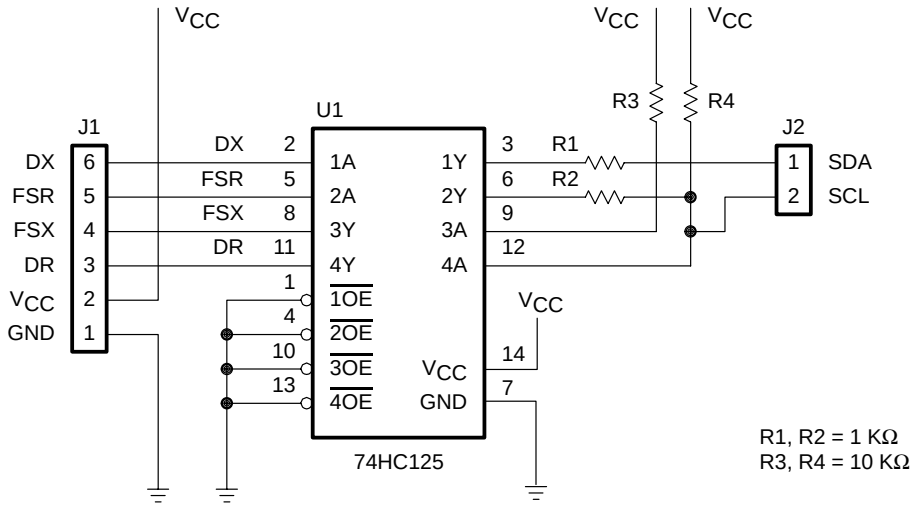
```
     RET
;***************************************************************************
; SDA = LOW ISR
;***************************************************************************
pullSDA:
     ST #PCR_SUB,*AR2+               ;Sub-bank address (SPSA0)=PCR0.
     ANDM  #K_DX_STAT_0,*AR2-        ;Mask off DX_STAT BIT.
     NOP
     NOP
     RET


;***************************************************************************
; SCL = HIGH ISR
;***************************************************************************
releaseSCL:
     ST #PCR_SUB,*AR2+               ;Sub-bank address (SPSA0)=PCR0.
     ORM #K_FSRP_1,*AR2-             ;Mask on FSR BIT.
     NOP
     NOP
     RET
;***************************************************************************
; SCL = LOW ISR
;***************************************************************************
pullSCL:
     ST #PCR_SUB,*AR2+               ;Sub-bank address (SPSA0)=PCR0.
     ANDM #K_FSRP_0,*AR2-            ;Mask off FSR BIT.
     NOP
     NOP
     RET
;***************************************************************************
;***************************************************************************

;***************************************************************************
; SET MCBSP AS GPIO
;***************************************************************************
_init_gpio:
     NOP
     PSHM  AR2
     NOP
     STM   #SPSA,AR2                 ;AR2 points to SPSA0.

     ST #SPCR2_SUB,*AR2+             ;Sub-bank address (SPSA0)=SPCR20.
     ST #0000000000000000b,*AR2-     ;
```

```
        ;~~~~~~~~~~~~~~~0 (XRST) ;GPIO : transmitter is reset

     ST #SPCR1_SUB,*AR2+ ;Sub-bank address (SPSA0)=SPCR20
     ST #0000000000000000b,*AR2- ;
        ;~~~~~~~~~~~~~~~0 (RRST)    ;GPIO : receiver set

     ST #PCR_SUB,*AR2+ ;Sub–bank address (SPSA0)=PCR0
     ST #0011010100100101b,*AR2- ;
        ;~~1~~~~~~~~~~~~ (XIOEN)   ;GPIO transmitter is enabled.
        ;~~~1~~~~~~~~~~~ (RIOEN)   ;GPIO receiver is enabled.
        ;~~~~0~~~~~~~~~~ (FSXM)    ;FSX set as input
        ;~~~~~1~~~~~~~~~ (FSRM)    ;FSR set as output
        ;~~~~~~~1~~~~~~~ (CLKRM)   ;CLKR set as output
        ;~~~~~~~~~~1~~~~~ (DX_STAT) ;Release SDA line.
        ;~~~~~~~~~~~~~1~~ (FSRP)    ;Release SCL line.
        ;~~~~~~~~~~~~~~~1 (CLKRP)   ;CLKR = 1.

     NOP                           ;McBSP takes 2 clock cycle
     NOP                           ;to start.
     POPM AR2
     NOP
     RET
;***************************************************************************
;***************************************************************************


;***************************************************************************
; INITIATE I2C
;***************************************************************************
_init_i2c:
     NOP
     PSHM AR1
     NOP
     STM #_I2CSTATUS,AR1
     ST #(K_RESET_I2C|K_CSDA_1|K_CSCL_1),*AR1+  ;I2CSTATUS
     ST #0,*AR1+               ;ODBYTECTR
     ST #7,*AR1+               ;ODBITCTR
     ST #0,*AR1+               ;IDBYTECTR
     ST #7,*AR1+               ;IDBITCTR
     ST #0,*AR1+              ;ERRORCODE
     ST #K_SLAVEADDR,*AR1    ;SLAVE_ADDR
     NOP
```

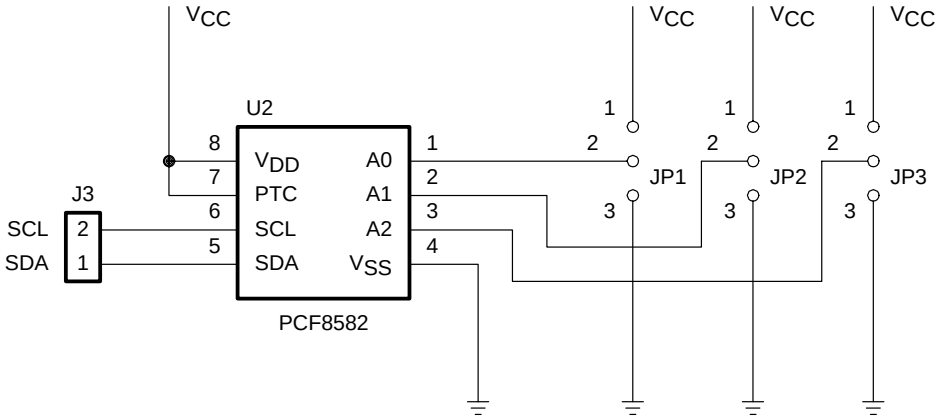
```
;***********************************************************************
; INIT TIMER
;***********************************************************************
     STM #STOP_TIMER,TCR       ;Stop timer.
     STM #K_PRD_VAL,PRD        ;Load PRD.
     STM #START_TIMER,TCR      ;Start timer.
     ORM #0008H,*(IMR)         ;Enable timer interrupt.
     STM #0FFFFH,IFR           ;Clear any pending interrupts.

     NOP
     POPM AR1
     NOP
     RET
;***********************************************************************
;***********************************************************************

;***********************************************************************
; INIT MASTER-TRANSMITTER PROCESS
;***********************************************************************
_write_i2c:
     NOP
     PSHM AR1
     PSHM AR2
     NOP
     STM #_I2CSTATUS,AR1       ;Init i2cstatus register for write.
     ST #(K_START_1|K_WRITE_TO_SLAVE|K_ADDR_1|K_CSDA_1|K_CSCL_1),*AR1
     NOP
     STM #_SLAVE_ADDR,AR1      ;1ST loc of ODPTR is slave address.
     LD *AR1,1,A               ;Shift slave address 1 place to the left.
     ADD #K_I2C_WRITE,A        ;Write mode of operation to LSB.
     STM #_ODPTR,AR2
     STL A,*AR2                ;Store back to _ODPTR.
     STM #_ODBYTECTR,AR1
     ADDM #1,*AR1              ;Add one more byte for slave address.
     NOP
     POPM AR2
     POPM AR1
     NOP
     RETE

;***********************************************************************
;***********************************************************************
```

```
;***************************************************************************
; INIT MASTER-RECEIVER PROCESS
;***************************************************************************
_read_i2c:
     NOP
     PSHM AR1
     NOP
     STM #_I2CSTATUS,AR1       ;Init i2cstatus register for read.
     ST #(K_START_1|K_READ_FROM_SLAVE|K_ADDR_1|K_CSDA_1|K_CSCL_1),*AR1
     NOP
     STM #_SLAVE_ADDR,AR1      ;1ST loc of ODPTR is slave address.
     LD *AR1,1,A               ;Shift slave address 1 place to the left.
     ADD #K_I2C_READ,A         ;Write mode of operation to LSB.
     STM #_ODPTR,AR1
     STL A,*AR1                ;Store back to _ODPTR.
     NOP
     STM #_ODBYTECTR,AR1       ;Only 1 byte(slave addr) to be sent.
     ST #2,*AR1                ;Must store 1+1 num of bytes to send

     NOP
     POPM AR1
     NOP
     RETE

;***************************************************************************
     .end

;***************************************************************************
;*******************END*OF*FILE*****************************************
;***************************************************************************
```

Appendix D Application Schematic



I2C interface circuit



PCF8582 EEPROM application

Figure D–1. Application Schematics of I2C Interface Circuit and PCF8582 EEPROM Application

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265